APPENDIX E

```
//
//  ------------------------------------------------------------------------
//  |----------------------------------------------------------------------|
//  |                                                                      |
//  |   FILE:   labelcl.cpp                                                |
//  |   FUNCTIONALITY: class Database definition                           |
//  |   PROGRAM: required to codes which opens database config file        |
//  |   COMMENTS: attempt to use symbols for patient ID that can           |
//  |             link to specific database, symbols for space             |
//  |             reduction representing files for retrieval               |
//  |   AUTHOR: A. CHRISTIAN TAHAN                                         |
//  |   DATE FIRST VERSION:  02/12/00                                      |
//  |----------------------------------------------------------------------|
//  ------------------------------------------------------------------------

#include "Labelcl.h"

Boolean LabelFileInterface::Configure(const String &  config_file,const
String &  section)
        {
        ConfigFile conf;
        String labeltype, label_extension;

        conf.Open_File(config_file);

        conf.Get_String_Opt("DBaseOptions", "LabelFileType", labeltype);

        conf.Get_String_Opt("DBaseOptions", "LabelFileExtension",
label_extension);

        //add here a new class for label handling

        if ( labeltype=="ntimitLabel")
                {
                New_For_Polimorphic_Pointer_With_Allocated_Class
(NTimitLabelClass);
                (*this)->label=NTimitLabel;
                }
                else
        if (labeltype=="ntimitreducedlabel")
                {
                New_For_Polimorphic_Pointer_With_Allocated_Class
(NTimitReducedLabelClass);
                (*this)->label=NTimitReducedLabel;
                }
                else
```

```
        if (labeltype=="ntimit39label")
                {
                New_For_Polimorphic_Pointer_With_Allocated_Class
(NTimit39LabelClass);
                (*this)->label=NTimit39Label;
                }
                else
        if (labeltype=="atislabel")
                {
                New_For_Polimorphic_Pointer_With_Allocated_Class
(AtisLabelsClass);
                (*this)->label=AtisLabel;
                }
                else
        if (labeltype=="atisreducedlabel")
                {
                New_For_Polimorphic_Pointer_With_Allocated_Class
(AtisReducedLabelsClass);
                (*this)->label=AtisReducedLabel;
                }
                else
        if (labeltype=="apascireducedlabel")
                {
                New_For_Polimorphic_Pointer_With_Allocated_Class
(ApasciReducedLabelsClass);
                (*this)->label=ApasciReducedLabel;
                }
                else
        if (labeltype=="apascilabel")
                {
                New_For_Polimorphic_Pointer_With_Allocated_Class
(ApasciLabelsClass);
                (*this)->label=ApasciLabel;
                }
                else
        if (labeltype=="LabelsFromFile")
                {
                New_For_Polimorphic_Pointer_With_Allocated_Class
(CustomLabelsFromFile);
                (*this)->label=LabelsFromFile;
                }

         else   {
                        merr<<"Unknown label file type specified in config
file: "<<labeltype;
                        }
```

```
        return ((*this)->Initialize(config_file,section, label_extension));
        }


//*****************************************************************
//                                                                 *
//                  LabelSymbolTable                               *
//                                                                 *
//*****************************************************************



void LabelSymbolTable::Translate_Symbol(String & symb, t_index num_sym)
const
        {
        symb=symb_table[num_sym];
        return;
        }

t_index LabelSymbolTable::Translate_Symbol(const String & sym) const
        {

        t_index num=0;
        t_index len_sym;

        len_sym=symb_table.Dim();

        while(num<len_sym AND symb_table[num]!=sym)
                num++;

        if(num==len_sym)
                {
                merr<<"symbol not found in Translate_Symbol: "<<sym;
                }

        return num;
        }


t_index LabelSymbolTable::Get_Num_Of_Symbols()  const
        {
        return symb_table.Dim();
        }
```

```
//*****************************************************************
//                                                                 *
//                    FileOfLabels                                    *
//                                                                 *
//*****************************************************************
FileOfLabels::FileOfLabels()
        {
        prev_entry=0;
        }
FileOfLabels::~FileOfLabels()
        {
        prev_entry=0;
        }

        if (symb_position[prev_entry].pos_in_file==0UL OR
                        symb_position[0].pos_in_file==0UL)
                merr<<"Unsegmented file. Filtered access not possible";

        if (symb_position[prev_entry].pos_in_file>act_smp AND
                        symb_position[prev_entry].num_sym==sym)
                {
                Assert(prev_entry==0 OR symb_position
[prev_entry-1].pos_in_file<act_smp);
                new_smp_pos=act_smp;
                return (Boolean)TRUE;
                }

        prev_entry++;
        while (prev_entry<symb_position.Dim() AND
                                        symb_position[prev_entry].num_sym!
=sym)
                        prev_entry++;

        if(prev_entry==symb_position.Dim())
                {
                new_smp_pos = 0;
                prev_entry = 0;
                return (Boolean)FALSE;
                }
                else    {
                                new_smp_pos=symb_position
[prev_entry-1].pos_in_file;
                                return (Boolean)TRUE;
                                }
```

```
        //BINARY search

        t_index step,i;
        i=(symb_position.Dim())/2;
        step=(1+i)/2;

        while(!(symb_position[i].pos_in_file<smp AND symb_position[i
+1].pos_in_file>smp)
                  AND !(symb_position[i].pos_in_file>smp AND i==0))
                {
                if (symb_position[i].pos_in_file<smp)
                        i+=step;
                else i-=step;

                step=(1+step)/2;

                }

        if(i!=0 OR symb_position[i].pos_in_file<smp)
                i++;
        sym=symb_position[i].num_sym;
        i=prev_entry;
        return;
        }


        for(j=0;j<dim;j++)
                {
                Translate_Symbol(tempsymb, symb_position[j].num_sym);
                file<<tempsymb<<" ";
                }
        file<<endl;
        return file;
        }


void GenericFileOfLabels::Reset()
        {
        label= No_Symbol;
        prev_entry=0;
        symb_position.Reset();
        symb_table.Reset();
        return;
        }

//*****************************************************************
```

```
//                                                              *
//                  NTimitLabel                                 *
//                                                              *
//**************************************************************
Boolean NTimitLabelClass::Initialize(const String &  file_name,
                                                    const String &
file_section, const String &label_ext)
        {
        t_index num_sym,i;

        label_extension=label_ext;

        Assert(label==NTimitLabel);

        const char *list_of_symbols[]=

{"iy","ih","eh","ae","ux","ix","ax","ah","uw","uh","ao","aa","ey","ay","oy"
,"aw",

"ow","l","r","y","w","er","axr","el","em","en","eng","m","n","ng","ch","jh"
,

"dh","b","d","dx","nx","g","p","t","k","q","z","zh","v","f","th","s",

"sh","hh","hv","pcl","tcl","kcl","qcl","bcl","dcl","gcl","epi","h#","#h","p
au","ax-h"};

        num_sym=63;

        symb_table.Destroy_And_ReDim(num_sym);
        for(i=0;i<num_sym;i++)
                  symb_table[i]=list_of_symbols[i];

        return TRUE;
        }


Boolean NTimitLabelClass::Open_Sym(const String & file_name)
        {

        String name,temp;
        ifstream f_lis;
        t_index i=0;
        t_index num_sym=0;
        t_index temp_num;
```

```
        prev_entry=0;
        name<<file_name<<"."<<label_extension;
        f_lis.open(name,ios::in|ios::nocreate);
        if(f_lis.fail())

        while (NOT f_lis.eof())
                {
                f_lis>>temp_num;
                f_lis>>temp_num;
                f_lis>>temp;
                if(!(f_lis.eof() AND temp[0]==EOF)) num_sym++;
                }


        f_lis.clear();
        f_lis.seekg(0,ios::beg);

        if (num_sym==0)
                merr<<"Empty file of ID transcription "<<name;

        symb_position.Destroy_And_ReDim(num_sym);
        for (i=0;i<num_sym;i++)
                {
                f_lis>>temp_num;
                f_lis>>symb_position[i].pos_in_file;
                f_lis>>temp;
                symb_position[i].num_sym=Translate_Symbol(temp);
                }

        f_lis.close();

        return TRUE;
        }


//*************************************************************
//                                                            *
//                  NTimitReducedLabel                          *
//                                                            *
//*************************************************************

Boolean NTimitReducedLabelClass::Open_Sym(const String & file_name)
        {

        String name,temp;
        ifstream f_lis;
```

```
        t_index i=0;
        t_index num_sym=0;
        t_index temp_num;

        prev_entry=0;
        name<<file_name<<"."<<label_extension;
        f_lis.open(name,ios::in|ios::nocreate);
        if(f_lis.fail())
                merr<<"Could not open ID transcription file: "<<name;


        while (NOT f_lis.eof())
                {
                f_lis>>temp_num;
                f_lis>>temp_num;
                f_lis>>temp;
                if(!(f_lis.eof() AND temp[0]==EOF)) num_sym++;
                }


        f_lis.clear();
        f_lis.seekg(0,ios::beg);

        if (num_sym==0)
                merr<<"Empty file of ID transcription "<<name;

        symb_position.Destroy_And_ReDim(num_sym);
        for (i=0;i<num_sym;i++)
                {
                f_lis>>temp_num;
                f_lis>>symb_position[i].pos_in_file;
                f_lis>>temp;
                symb_position[i].num_sym=Translate_Symbol(temp);
                }

        f_lis.close();

        return TRUE;
        }

t_index NTimitReducedLabelClass::Translate_Symbol(const String & sym) const
        {
        t_index num=0;
        t_index len_sym;

        Assert(label == NTimitReducedLabel OR label ==AtisLabel);
```

```
        len_sym=symb_table.Dim();
        while(num<len_sym AND symb_table[num]!=sym)
                num++;

        if(num==len_sym)
                {
                if(sym=="ux") num=7;   else
                if(sym=="el") num=16;  else
                if(sym=="axr") num=20; else
                if(sym=="ax-h") num=5; else
                if(sym=="em") num=21;  else
                if(sym=="en") num=22;  else
                if(sym=="nx") num=22;  else
                if(sym=="eng") num=23; else
                if(sym=="q") num=32;   else
                if(sym=="hv") num=40;  else
                if(sym=="pcl") num=41; else
                if(sym=="tcl") num=41; else
                if(sym=="kcl") num=41; else
                if(sym=="qcl") num=41; else
                if(sym=="bcl") num=42; else
                if(sym=="dcl") num=42; else
                if(sym=="gcl") num=42; else
                if(sym=="***") num=100; else  //separator

                if(sym=="#h" OR sym=="h#" OR sym == "pau" )
                        num=44;
                else {
                         merr<<"unknown symbol of NTIMIT. Symbol: "<<sym;
                         }
                }


        return num;
        }


Boolean NTimitReducedLabelClass::Initialize(const String & file_name,

const String & section_name, const String &label_ext)
        {
        t_index num_sym,i;

        label_extension=label_ext;
```

```
        Assert(label==NTimitReducedLabel);

        const char *list_of_symbols[]=

{"iy","ih","eh","ae","ix","ax","ah","uw","uh","ao","aa","ey","ay","oy","aw"
,"ow",

"l","r","y","w","er","m","n","ng","ch","jh","dh","b","d","g","p","t",

"k","z","zh","v","f","th","s","sh","hh","cl","vcl","epi","sil","dx"};

        num_sym=46;

        symb_table.Destroy_And_ReDim(num_sym);
        for(i=0;i<num_sym;i++)
                  symb_table[i]=list_of_symbols[i];

        return TRUE;
        }

//*****************************************************************
//                                                                 *
//                  NTimit39Label                                  *
//                                                                 *
//*****************************************************************
Boolean NTimit39LabelClass::Open_Sym(const String & file_name)
        {

        String name,temp;
        ifstream f_lis;
        t_index i=0;
        t_index num_sym=0;
        t_index temp_num;

        prev_entry=0;
        name<<file_name<<"."<<label_extension;
        f_lis.open(name,ios::in|ios::nocreate);
        if(f_lis.fail())
                merr<<"Could not open ID transcription file: "<<name;


        while (NOT f_lis.eof())
                {
                f_lis>>temp_num;
                f_lis>>temp_num;
                f_lis>>temp;
```

```
                if(!(f_lis.eof() AND temp[0]==EOF)) num_sym++;
                }


        f_lis.clear();
        f_lis.seekg(0,ios::beg);

        if (num_sym==0)
                merr<<"Empty file of ID transcription "<<name;

        symb_position.Destroy_And_ReDim(num_sym);
        for (i=0;i<num_sym;i++)
                {
                f_lis>>temp_num;
                f_lis>>symb_position[i].pos_in_file;
                f_lis>>temp;
                symb_position[i].num_sym=Translate_Symbol(temp);
                }

        f_lis.close();

        return TRUE;
        }

t_index NTimit39LabelClass::Translate_Symbol(const String & sym) const
        {
        t_index num=0;
        t_index len_sym;

        Assert(label == NTimit39Label);

        len_sym=symb_table.Dim();
        while(num<len_sym AND symb_table[num]!=sym)
                num++;

        if(num==len_sym)
                {
                if(sym=="#h" OR sym=="h#" OR sym == "pau" OR sym == "pcl"
                                OR sym == "tcl" OR sym == "kcl" OR sym == "bcl"
OR sym == "dcl"
                                OR sym == "gcl" OR sym == "qcl" OR sym == "epi")
num=37; //sil

                else if(sym=="ux") num=5;                //uw
                else if(sym=="el") num=13;               // l
                else if(sym=="axr") num=17;              //er
```

```
                else if(sym=="ax-h" OR sym=="ax") num=4; //ah
                else if(sym=="em") num=18;               // m
                else if(sym=="en" OR sym=="nx") num=19;  // n
                else if(sym=="eng") num=20;              //ng
                else if(sym=="q") num=29;                                    //
k
                else if(sym=="hv") num=36;                                   //
hh
                else if(sym=="ao") num=7;                                    //
aa
                else if(sym=="ix") num=1;                                    //
ih
                else if(sym=="zh") num=35;                                   //
sh
                else if(sym=="***") num=100;                        //
separator

                else {
                        merr<<"unknown symbol of NTIMIT. Symbol: "<<sym;
                        }
        }

        return num;
        }


Boolean NTimit39LabelClass::Initialize(const String & file_name,

        const String & section_name, const String &label_ext)
        {
        t_index num_sym,i;

        label_extension=label_ext;

        Assert(label==NTimit39Label);

        const char *list_of_symbols[]=

{"iy","ih","eh","ae","ah","uw","uh","aa","ey","ay","oy","aw","ow",

"l","r","y","w","er","m","n","ng","ch","jh","dh","b","d","g",
                "p","t","k","z","v","f","th","s","sh","hh","sil","dx"};

        num_sym=39;

        symb_table.Destroy_And_ReDim(num_sym);
```

```
        for(i=0;i<num_sym;i++)
                symb_table[i]=list_of_symbols[i];

        return TRUE;
        }



Boolean AtisReducedLabelsClass::Initialize(const String &  file_name,
                                                                const
String &  section_name, const String &label_ext)
        {
        t_index num_sym,i;

        label_extension=label_ext;

        Assert(label== AtisReducedLabel);

        const char *list_of_symbols[]=

{"iy","ih","eh","ae","ah","uw","uh","aa","ey","ay","oy","aw","ow",
                 "l","r","y","w","m","n","ch","jh","dh","b","d","g",
                 "p","t","k","z","v","f","th","s","sh","hh","sil"};

        num_sym=36;


        symb_table.Destroy_And_ReDim(num_sym);

        for(i=0;i<num_sym;i++)
                symb_table[i]=list_of_symbols[i];

        return TRUE;
        }

Boolean AtisReducedLabelsClass::Open_Sym(const String &file_name)
{
        String name,temp;
        ifstream f_lis;
        t_index i=0;
        t_index num_sym=0;
        t_index temp_num;

        prev_entry=0;
        name<<file_name<<"."<<label_extension;
```

```
        f_lis.open(name,ios::in|ios::nocreate);
        if(f_lis.fail())
                merr<<"Could not open ID transcription file: "<<name;


        while (NOT f_lis.eof())
                {
                f_lis>>temp_num;
                f_lis>>temp_num;
                f_lis>>temp;
                if(!(f_lis.eof() AND temp[0]==EOF)) num_sym++;
                }


        f_lis.clear();
        f_lis.seekg(0,ios::beg);

        if (num_sym==0)
                merr<<"Empty file of ID transcription "<<name;

        symb_position.Destroy_And_ReDim(num_sym);
        for (i=0;i<num_sym;i++)
                {
                f_lis>>temp_num;
                f_lis>>symb_position[i].pos_in_file;
                f_lis>>temp;
                symb_position[i].num_sym=Translate_Symbol(temp);
                }


        f_lis.close();

        return TRUE;
        }

t_index AtisReducedLabelsClass::Translate_Symbol(const String & sym) const
        {
        t_index num=0;
        t_index len_sym;

        Assert(label ==AtisReducedLabel);

        len_sym=symb_table.Dim();
        while(num<len_sym AND symb_table[num]!=sym)
                num++;
```

```
        if(num==len_sym)
                {
                        if(sym=="ao")  num=7;    else // aa
                        if(sym=="ix")  num=1;    else // ih
                        if(sym=="nx")  num=18;   else // n
                        if(sym=="ax")  num=4;    else // ah
                        if(sym=="zh")  num=33;   else // sh
                        if(sym=="***") num=100;  else // separator

                                {
                                merr<<"unknown symbol of ATISLabel. Symbol:
"<<sym;
                                }
                }

        return num;

        }



Boolean AtisLabelsClass::Open_Sym(const String & file_name)
{

        String name,temp;
        ifstream f_lis;
        t_index i=0;
        t_index num_sym=0;

        prev_entry=0;
        name<<file_name<<"."<<label_extension;
        f_lis.open(name,ios::in|ios::nocreate);
        if(f_lis.fail())
                merr<<"Could not open ID transcription file: "<<name;

        while (!f_lis.eof())
                {
                f_lis>>temp;
                if(!(f_lis.eof() AND temp[0]==EOF)) num_sym++;
                }
        f_lis.clear();
        f_lis.seekg(0,ios::beg);

        if (num_sym==0)
                merr<<"Empty file of ID transcription "<<name;
```

```
        symb_position.Destroy_And_ReDim(num_sym);
        for (i=0;i<num_sym;i++)
                {
                f_lis>>temp;
                symb_position[i].num_sym=Translate_Symbol(temp);
                }

        f_lis.close();

        return TRUE;
        }


Boolean AtisLabelsClass::Initialize(const String &  file_name,
                                                                const
String &  section_name, const String &label_ext)
        {
        t_index num_sym,i;

        label_extension=label_ext;

        Assert(label== AtisLabel);
        {
        const char *list_of_symbols[]=

{"iy","ih","ix","eh","ae","ao","ax","uw","uh","aa","ey","ay","oy","aw","ow"
,

"l","r","y","w","er","m","n","ng","nx","ch","jh","dh","b","d","g",
                "p","t","k","z","v","f","th","s","sh","zh","hh","sil"};


        num_sym=42;

        symb_table.Destroy_And_ReDim(num_sym);

        for(i=0;i<num_sym;i++)
        symb_table[i]=list_of_symbols[i];

        }

        return TRUE;
        }

Boolean ApasciLabelsClass::Open_Sym(const String & file_name)
{
```

```
        String name,temp;
        ifstream f_lis;
        t_index i=0;
        t_index num_sym=0;
        t_index temp_num;

        prev_entry=0;
        name<<file_name<<"."<<label_extension;
        f_lis.open(name,ios::in|ios::nocreate);
        if(f_lis.fail())
                merr<<"Could not open ID transcription file: "<<name;


        while (NOT f_lis.eof())
                {
                f_lis>>temp_num;
                f_lis>>temp_num;
                f_lis>>temp;
                if(!(f_lis.eof() AND temp[0]==EOF)) num_sym++;
                }


        f_lis.clear();
        f_lis.seekg(0,ios::beg);

        if (num_sym==0)
                merr<<"Empty file of ID transcription "<<name;

        symb_position.Destroy_And_ReDim(num_sym);
        for (i=0;i<num_sym;i++)
                {
                f_lis>>temp_num;
                f_lis>>symb_position[i].pos_in_file;
                f_lis>>temp;
                symb_position[i].num_sym=Translate_Symbol(temp);
                }

        f_lis.close();

        return TRUE;
        }

t_index ApasciLabelsClass::Translate_Symbol(const String & sym) const
        {
        t_index num=0;
        t_index len_sym;
```

```
        Assert(label == ApasciLabel);

        len_sym=symb_table.Dim();
        while(num<len_sym AND symb_table[num]!=sym)
                num++;

        if(num==len_sym)
                {
                if(sym=="E") num=1;                                     // e
                else if(sym=="O") num=3;                 // o
                else if(sym=="@bg") num=48;                                     //
sil
                else if(sym=="***") num=100;                            //
separator
                else {
                        merr<<"unknown symbol of APASCI Symbol: "<<sym;
                        }
        }

        return num;
        }


Boolean ApasciLabelsClass::Initialize(const String &  file_name,
                                                                const
String &  section_name, const String &label_ext)
        {
        t_index num_sym,i;

        label_extension=label_ext;

        Assert(label== ApasciLabel);

        const char *list_of_symbols[]=
                {"a","e","i","o","u","f","v","s","z","S","ff","vv","ss",

"SS","tS","dZ","ts","dz","ttS","ddZ","tts","ddz","j","w","p","t","k","b",

"d","g","pp","tt","kk","bb","dd","gg","m","n","J","mm","nn","JJ","l","r",
                 "L","ll","rr","LL","sil", "@sch"};


        num_sym=50;

        symb_table.Destroy_And_ReDim(num_sym);
```

```
        for(i=0;i<num_sym;i++)
        symb_table[i]=list_of_symbols[i];

        return TRUE;
        }


Boolean ApasciReducedLabelsClass::Open_Sym(const String & file_name)
        {

        String name,temp;
        ifstream f_lis;
        t_index i=0;
        t_index num_sym=0;
        t_index temp_num;

        prev_entry=0;
        name<<file_name<<"."<<label_extension;
        f_lis.open(name,ios::in|ios::nocreate);
        if(f_lis.fail())
                merr<<"Could not open ID transcription file: "<<name;


        while (NOT f_lis.eof())
                {
                f_lis>>temp_num;
                f_lis>>temp_num;
                f_lis>>temp;
                if(!(f_lis.eof() AND temp[0]==EOF)) num_sym++;
                }


        f_lis.clear();
        f_lis.seekg(0,ios::beg);

        if (num_sym==0)
                merr<<"Empty file of ID transcription "<<name;

        symb_position.Destroy_And_ReDim(num_sym);
        for (i=0;i<num_sym;i++)
                {
                f_lis>>temp_num;
                f_lis>>symb_position[i].pos_in_file;
                f_lis>>temp;
                symb_position[i].num_sym=Translate_Symbol(temp);
```

```
                }

        f_lis.close();

        return TRUE;
        }

t_index ApasciReducedLabelsClass::Translate_Symbol(const String & sym)
const
        {
        t_index num=0;
        t_index len_sym;

        Assert(label == ApasciReducedLabel);

        len_sym=symb_table.Dim();
        while(num<len_sym AND symb_table[num]!=sym)
                num++;

        if(num==len_sym)
                {
                if(sym=="E") num=1;                             // e
                else if(sym=="O") num=3;                // o
                else if(sym=="ff") num=5;                   // f
                else if(sym=="vv") num=6;                               //
v
                else if(sym=="ss") num=7;               // s
                else if(sym=="SS") num=9;                               //
S
                else if(sym=="ttS") num=10;             //tS
                else if(sym=="ddZ") num=11;                             //
dZ
                else if(sym=="tts") num=12;                             //
ts
                else if(sym=="ddz") num=13;                             //
dz
                else if(sym=="pp") num=16;                              //
p
                else if(sym=="tt") num=17;                              //
t
                else if(sym=="kk") num=18;                          // k
                else if(sym=="bb") num=19;                          // b
                else if(sym=="dd") num=20;                          // d
                else if(sym=="gg") num=21;                          // g
                else if(sym=="mm") num=22;                          // m
                else if(sym=="nn") num=23;                          // n
```

```
                else if(sym=="JJ") num=24;                              // J
                else if(sym=="ll") num=25;                              // l
                else if(sym=="rr") num=26;                              // r
                else if(sym=="LL") num=27;                              // L
                else if(sym=="@bg") num=28;                                 //
sil
                else if(sym=="***") num=100;                        //
separator
                else {
                        merr<<"unknown symbol of APASCI Symbol: "<<sym;
                        }
        }

        return num;
        }


Boolean ApasciReducedLabelsClass::Initialize(const String & file_name,

        const String & section_name, const String &label_ext)
        {
        t_index num_sym,i;

        label_extension=label_ext;

        Assert(label==ApasciReducedLabel);

        const char *list_of_symbols[]=

{"a","e","i","o","u","f","v","s","z","S","tS","dZ","ts","dz","j","w","p",
                 "t","k","b","d","g","m","n","J","l","r","L","sil","@sch"};

        num_sym=30;

        symb_table.Destroy_And_ReDim(num_sym);
        for(i=0;i<num_sym;i++)
                symb_table[i]=list_of_symbols[i];

        return TRUE;
        }



//*************************************************************
//                                                            *
//                  LabelTrans                          *
```

```
//                                                                *
//****************************************************************

Boolean CustomLabelsFromFile::Initialize(const String &  file_name,

                const String &  section_name, const String &label_ext)
        {
        merr<<"this function must be implemented";
        return TRUE;
        }

Boolean CustomLabelsFromFile::Open_Sym(const String & file_name)
        {
        merr<<"this function must be implemented";
        return TRUE;
        }
```